InterviewBit

NumPY Interview Questions

To view the live version of the page, click here.

Contents

NumPY Interview Questions for Freshers

NumPY Interview Questions for Experienced

NumPY Interview Questions for Experienced

(.....Continued)

19. What happens when the split() method is used for splitting NumPy arrays?

20. What happens when we use the arrays_split() method for splitting the NumPy array?

21. How will you implement the moving average for the 1D array in NumPy?

22. How is fliplr different from flipud methods in NumPy?

NumPy Coding Questions

23. Write a program to convert a string element to uppercase, lowercase, capitalise the first letter, title-case and swapcase of a given NumPy array.

24. Write a program to transform elements of a given string to a numeric string of 10 digits by making all the elements of a given string to a numeric string of 8 digits with zeros on the left.

25. Write a program for inserting space between characters of all elements in a NumPy array.

26. Write a program to repeat each of the elements five times for a given array.

27. Write a program for creating an integer array with values belonging to the range 10 and 60

28. Write a program to add a border of zeros around the existing array.

29. Write a program for changing the dimension of a NumPy array.

30. Write a program for interchanging two axes of the NumPy array.

Let's get Started

NumPy is a python-based, open-source, powerful package used majorly for array processing. It is well-known for its tools that have very high performance and high efficiency while operating with N-dimensional powerful array objects. It is also used for performing scientific computations, and mathematical and statistical operations along with various broadcasting abilities. It is easy to learn and due to its powerful tools for processing millions of elements, it has become very popular among data scientists for developing various machine learning algorithms too.

Due to its popularity among the data science community, interviewers expect the **data scientist** candidates and the **python developers** to know about the NumPy package. In this article, we will see the most commonly asked NumPy interview questions and answers.

NumPY Interview Questions for Freshers

1. What is NumPy? Why should we use it?

NumPy (also called Numerical Python) is a highly flexible, optimized, open-source package meant for array processing. It provides tools for delivering high-end performance while dealing with N-dimensional powerful array objects. It is also beneficial for performing scientific computations, mathematical, and logical operations, sorting operations, I/O functions, basic statistical and linear algebra-based operations along with random simulation and broadcasting functionalities. Due to the vast range of capabilities, NumPy has become very popular and is the most preferred package. The following image represents the uses of NumPy.

Uses of NumPy

2. Why is NumPy preferred over Matlab, Octave, Idl or Yorick?

NumPy is an open-source, high-performing library that allows complex mathematical and scientific computational capabilities. It makes use of Python language which is a high-level, easy-to-learn, general-purpose programming language. It supports the following:

- Powerful functions for performing complex mathematical operations on multi-dimensional matrices and arrays. The operations on ndarrays of NumPy are approximately up to 50% faster when compared to operations on native lists using loops. This efficiency is very much useful when the arrays have millions of elements.
- Provides indexing syntax to access portions of data easily in a large array.
- Provides built-in functions which help to easily perform operations related to linear algebra and statistics.
- It takes only a few lines of code to achieve complex computations using NumPy.

3. How are NumPy arrays better than Python's lists?

- Python lists support storing heterogeneous data types whereas NumPy arrays can store datatypes of one nature itself. NumPy provides extra functional capabilities that make operating on its arrays easier which makes NumPy array advantageous in comparison to Python lists as those functions cannot be operated on heterogeneous data.
- NumPy arrays are treated as objects which results in minimal memory usage. Since Python keeps track of objects by creating or deleting them based on the requirements, NumPy objects are also treated the same way. This results in lesser memory wastage.
- NumPy arrays support multi-dimensional arrays.
- NumPy provides various powerful and efficient functions for complex computations on the arrays.
- NumPy also provides various range of functions for BitWise Operations, String Operations, Linear Algebraic operations, Arithmetic operations etc. These are not provided on Python's default lists.

4. What are ndarrays in NumPy?

ndarray object is the core of the NumPy package. It consists of n-dimensional arrays storing elements of the same data types and also has many operations that are done in compiled code for optimised performance. These arrays have fixed sizes defined at the time of creation. Following are some of the properties of ndarrays:

- When the size of ndarrays is changed, it results in a new array and the original array is deleted.
- The ndarrays are bound to store homogeneous data.
- They provide functions to perform advanced mathematical operations in an efficient manner.

5. What are ways of creating 1D, 2D and 3D arrays in NumPy?

Consider you have a normal python list. From this, we can create NumPy arrays by making use of the array function as follows:

- One-Dimensional array

```
import numpy as np

arr = [1,2,3,4]          #python list
numpy_arr = np.array(arr)     #numpy array
```

- Two-Dimensional array

```
import numpy as np

arr = [[1,2,3,4],[4,5,6,7]]
numpy_arr = np.array(arr)
```

- Three-Dimensional array

```
import numpy as np

arr = [[[1,2,3,4],[4,5,6,7],[7,8,9,10]]]
numpy_arr = np.array(arr)
```

Using the np.array() function, we can create NumPy arrays of any dimensions.

6. How do you find the data type of the elements stored in the NumPy arrays?

NumPy supports the following datatypes:

- i - integer
- S - string
- b - boolean
- f - float
- u - unsigned integer
- c - complex float
- m - timedelta
- M - datetime
- O - object
- U - unicode string
- V - fixed memory chunk for types such as void
  We can make use of the dtype property that returns the type of the elements stored in the NumPy array. Let us consider the below code snippet. We create some sample arrays and we see what the data types of these arrays are.

```python
import numpy as np

arr1 = np.array([1, 2, 3, 4])
arr2 = np.array(['I', 'love', 'Interviewbit'])       # Stored as Unicode characters with l
arr3 = np.array([1, 2, 3, 4], dtype='S')             # Creating numpy array of defined type

print(arr1.dtype)
print(arr2.dtype)
print(arr3.dtype)
```

The output will be:

```
int64
<U12
|S1
```

7. How can you reverse a NumPy array?

There are two ways of reversing a NumPy array.

- **Method 1:** Using the slicing method: We can make use of [::-1] for reversing the array. The following example demonstrates this:

```
import numpy as np

# create numpy array
arr = np.array([1, 2, 4, 6])

# To reverse array
reverse_arr = arr[::-1]
print(reverse_arr)
```

Output:

```
[6 4 2 1]
```

- **Method 2:** flipud function: This function is provided by NumPy to reverse the NumPy array. Let us see the below example about its usage.

```
import numpy as np

# create numpy array
arr = np.array([1, 2, 4, 5, 6])

#flipud method for reversing
reverse_arr = np.flipud(arr)
print(reverse_arr)
```

Output:

```
[6 5 4 2 1]
```

8. How is np.mean() different from np.average() in NumPy?

- np.mean() method calculates the arithmetic mean and provides additional options for input and results. For example, it has the option to specify what data types have to be taken, where the result has to be placed etc.
- np.average() computes the weighted average if the weights parameter is specified. In the case of weighted average, instead of considering that each data point is contributing equally to the final average, it considers that some data points have more weightage than the others (unequal contribution).

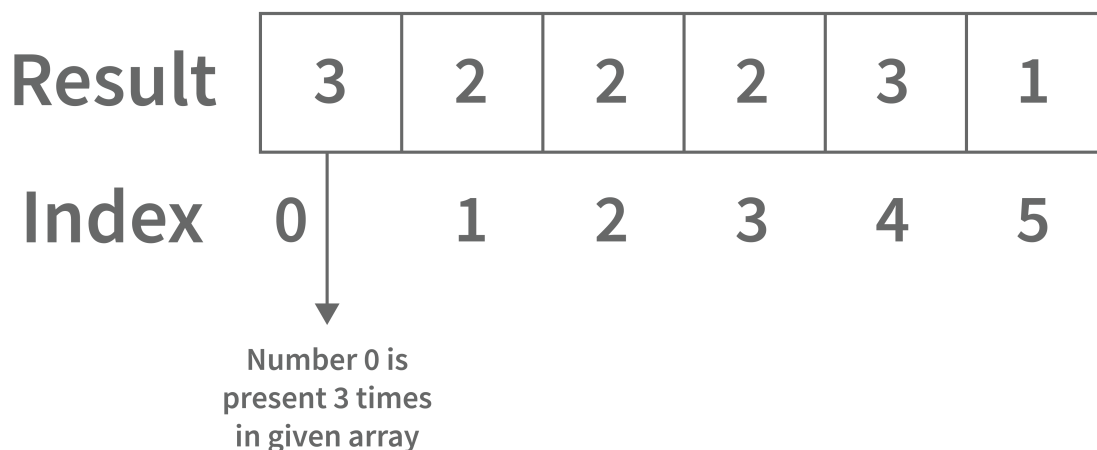## 9. How do you count the frequency of a given positive value appearing in the NumPy array?

We can make use of the bincount() function to compute the number of times a given value is there in the array. This function accepts only positive integers and boolean expressions as the arguments.

```
import numpy as np
arr = np.array([1, 2, 1, 3, 5, 0, 0, 0, 2, 3])
result = np.bincount(arr)
print(result)
```

The result is:

```
[3 2 2 2 0 1]
```

It has to be noted here that each element represents the count of the corresponding index value present in the original array. This is demonstrated in the below image:

Result	3	2	2	2	3	1

| Index | 0 | 1 | 2 | 3 | 4 | 5 |

Number 0 is
present 3 times
in given array

InterviewBit

10. How do we check for an empty array (or zero elements array)?

We can check for the emptiness of a NumPy array by making use of the size attribute. Let us consider the below example. We have NumPy array arr filled with zeros. If the size element returns zero, that means the array is empty or it only consists of zeros.

```
import numpy as np
arr = np.zeros((1,0))     #returns empty array
print(arr.size)          #returns 0
```

This return 0

11. How is arr[:,0] different from arr[:,[0]]

`arr[:,0]` - Returns 0th index elements of all rows. In other words, return the first column elements.

```
 import numpy as np

arr = np.array([[1,2,3,4],[5,6,7,8]])
new_arr =arr[:,0]
print(new_arr)
```

Output:

```
 [1 5]
```

`arr[:,[0]]` - This returns the elements of the first column by adding extra dimension to it.

```
import numpy as np

arr = np.array([[1,2,3,4],[5,6,7,8]])
new_arr =arr[:,[0]]
print(new_arr)
```

Output:

```
[[1]
 [5]]
```

12. How do you multiply 2 NumPy array matrices?

We can make use of the dot() for multiplying matrices represented as NumPy arrays. This is represented in the code snippet below:

```python
import numpy as np

# NumPy matrices
A = np.arange(15,24).reshape(3,3)
B = np.arange(20,29).reshape(3,3)
print("A: ",A)
print("B: ",B)

# Multiply A and B
result = A.dot(B)
print("Result: ", result)
```

Output

```
A:  [[15 16 17]
 [18 19 20]
 [21 22 23]]
B:  [[20 21 22]
 [23 24 25]
 [26 27 28]]
Result:  [[1110 1158 1206]
 [1317 1374 1431]
 [1524 1590 1656]]
```

13. How do you concatenate 2 NumPy arrays?

Concatenating 2 arrays by adding elements to the end can be achieved by making use of the concatenate() method of the NumPy package. Syntax:

```python
np.concatenate((a1, a2, ...), axis=0, out=None)
```

where,

- a1,a2: arrays of the same shape
- axis: Represents the axis along which the arrays are joined. The default value is 0.
- out: If mentioned, it specifies the destination for placing the result.

For example:

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])

# Concatenate with axis 0
c = np.concatenate((a,b), axis=0)
print("With axis 0: \n",c )

# Concatenate with axis 1 (b.T represents transpose matrix)
d = np.concatenate((a,b.T), axis=1)
print("With axis 1: \n",d )
```

The output would be:

```
With axis 0:
 [[1 2]
 [3 4]
 [5 6]]
With axis 1:
 [[1 2 5]
 [3 4 6]]
```

Notice how the arrays are concatenated with different values of the axis.

14. How do you convert Pandas DataFrame to a NumPy array?

The to_numpy() method of the NumPy package can be used to convert Pandas DataFrame, Index and Series objects.

Consider we have a DataFrame df, we can either convert the whole Pandas DataFrame df to NumPy array or even select a subset of Pandas DataFrame to NumPy array by using the to_numpy() method as shown in the example below:

```
import pandas as pd
import numpy as np
# Pandas DataFrame
df = pd.DataFrame(data={'A': [3, 2, 1], 'B': [6,5,4], 'C': [9, 8, 7]},
                  index=['i', 'j', 'k'])
print("Pandas DataFrame: ")
print(df)

# Convert Pandas DataFrame to NumPy Array
np_arr = df.to_numpy()
print("Pandas DataFrame to NumPy array: ")
print(np_arr)


# Convert specific columns of Pandas DataFrame to NumPy array
arr = df[['B', 'C']].to_numpy()
print("Convert B and C columns of Pandas DataFrame to NumPy Array: ")
print (arr)
```

The output of the above code is

```
Pandas DataFrame:
   A  B  C
i  3  6  9
j  2  5  8
k  1  4  7
Pandas DataFrame to NumPy array:
[[3 6 9]
 [2 5 8]
 [1 4 7]]
Convert B and C columns of Pandas DataFrame to NumPy Array:
[[6 9]
 [5 8]
 [4 7]]
```

NumPY Interview Questions for Experienced

15. What do you understand by Vectorization in NumPy?

Function Vectorization technically means that the function is applied to all elements in the array. Typically, certain python functionalities on arrays (such as loops) are slower in nature because python arrays can contain elements of different data types. Since the C program expects a specific datatype, there are chances of compiler optimisation which makes C code run faster. Since NumPy arrays support storing elements of a single datatype, most of the implementations of the functions written in NumPy meant for arithmetic, logical operations etc have optimised C program code under their hood. Additionally, NumPy also helps developers create their own vectorised functions by following the below steps:

- Write your required function that takes array elements as parameters.
- Vectorize the function by making use of the vectorize() method of the NumPy package.
- Give array inputs to the vectorized function.

The below example demonstrates the process of vectorization.

```python
import numpy as np
# Define your function
def add(arr1, arr2):
    return (arr1 + arr2)

arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])

#vectorize add method
vectorized_add = np.vectorize(add)

#call vectorized method
result = vectorized_add(arr1, arr2)

print(result)
```

The output of above code

```
[5 7 9]
```

16. How is vstack() different from hstack() in NumPy?

Both methods are used for combining the NumPy arrays. The main difference is that the hstack method combines arrays horizontally whereas the vstack method combines arrays vertically.

For example, consider the below code.

```
import numpy as np
a = np.array([1,2,3])
b = np.array([4,5,6])

# vstack arrays
c = np.vstack((a,b))
print("After vstack: \n",c)
# hstack arrays
d = np.hstack((a,b))
print("After hstack: \n",d)
```

The output of this code would be:

```
After vstack:
[[1 2 3]
 [4 5 6]]
After hstack:
[1 2 3 4 5 6]
```

Notice how after the vstack method, the arrays were combined vertically along the column and how after the hstack method, the arrays were combined horizontally along the row.

17. How do you find the local peaks (or maxima) in a 1-D NumPy Array?

Peaks are the points that are surrounded by smaller value points on either side as shown in the image below:
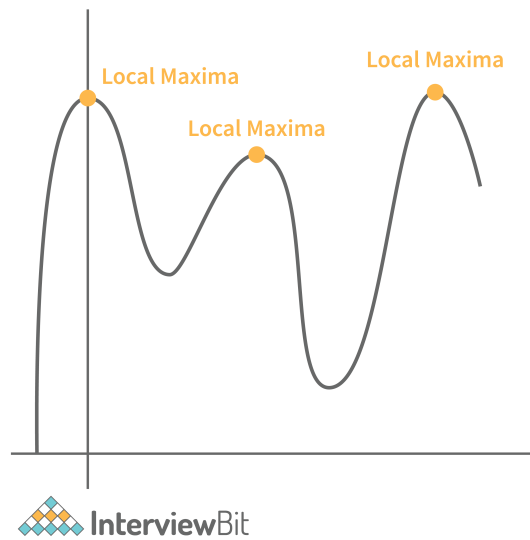
There are two ways of finding local maxima:

Using .where() method: This method lists all positions/indices where the element value at position i is greater than the element on either side of it. This method does not check for the points that have only one neighbour. This is demonstrated in the example below:

```
import numpy as np
# define NumPy array
arr = np.array([1, 4, 8, 1, 3, 5, 1, 6, 1, -5, -1, 19, 2])


maxima_peaks_positions = np.where((arr[1:-1] > arr[0:-2]) * (arr[1:-1] > arr[2:]))[0] +
print(maxima_peaks_positions)
```

Output:

```
[ 2  5  7 11]]
```

- The `+1` at the end of the expression is required as it finds the indexes within the slice `arr[1:-1]` and not the entire array arr.
- The where() method returns a tuple of arrays where the first element is our required array. Hence we add `[0]` after the where method.

Using combination of .diff(), .sign() and .where() method:

- In this method, we calculate the difference between each element using the diff() method of NumPy.
- Then we use the sign() method on the array to get the sign of difference.
- The value can be either -1 or +1. This result is then passed on to another diff() method which returns 0, -2 or +2 value. The value 0 indicates that the points are continuously increasing or decreasing, +2 indicates minimum peak and -2 indicates maximum peak (local maxima).
- We then identify the position or indexes of the local maxima using the where() method. The reason for using +1 at the end of where and [0] after where is the same as the explanation described in Method 1 for finding local maxima.

The following code example demonstrates this:

```
import numpy as np
# define NumPy array
arr = np.array([1, 4, 8, 1, 3, 5, 1, 6, 1, -5, -1, 19, 2])

all_peaks = np.diff(np.sign(np.diff(arr)))
maxima_peaks_positions = np.where(all_peaks == -2)[0] + 1
print(maxima_peaks_positions)
```

Output:

```
[ 2  5  7 11]]
```

18. How is Vectorization related to Broadcasting in NumPy?

Vectorization involves delegating NumPy operations internally to optimized C language functions to result in faster Python code. Whereas Broadcasting refers to the methods that allow NumPy to perform array-related arithmetic operations. The size or shape of the arrays does not matter in this case. Broadcasting solves the problem of mismatched shaped arrays by replicating the smaller array along the larger array to ensure both arrays are having compatible shapes for NumPy operations. Performing Broadcasting before Vectorization helps to vectorize operations which support arrays of different dimensions.

19. What happens when the split() method is used for splitting NumPy arrays?

1. np.split() : Equally splits arrays into multiple sub-arrays. It raises Value Error when the split cannot be equal.

- Syntax:

```
np.split(array, sections, axis=0)
```

where,

- array - array that needs to be split
- sections -
  - If we give an integer X, X equal sub-arrays are obtained after dividing the array. If the split is not possible, ValueError is raised.
    - For example:

```
 import numpy as np
a = np.arange(8)
split_arr = np.split(a, 2)
split_arr
```

Output

```
[array([0, 1, 2, 3]), array([4, 5, 6, 7])]
```

- If we give a 1-D sorted array then the entries would represent where the array would be split along the axis. For instance if we provide [2:3] and axis as 0, then the result would be

  `[arr[0:2], arr[2:3], arr[3:]]`

  - If the provided index exceeds the array dimension along the given axis, then an empty subarray will be returned.
  - For example:

```
[array([0., 1., 2.]),
 array([3.]),
 array([4.]),
 array([5.]),
 array([], dtype=float64),
 array([], dtype=float64),
 array([], dtype=float64)]
```

The output would be:

```
import numpy as np
a = np.arange(6.0)
split_arr = np.split(a, [3, 4, 5, 6, 7,8])
split_arr
```

- axis - Along what axis the array has to be split. By default, the value is 0

20. What happens when we use the arrays_split() method for splitting the NumPy array?

The array_split() method is similar to the split() method as it helps in splitting a given array to multiple subarrays. The main difference is that the array_split() allows sections to be an integer which does not result in equal array division. For an array of length L, if we want it to split to N subarrays, then L % N subarrays of size (L//N + 1) and remaining subarrays are of size L//N.
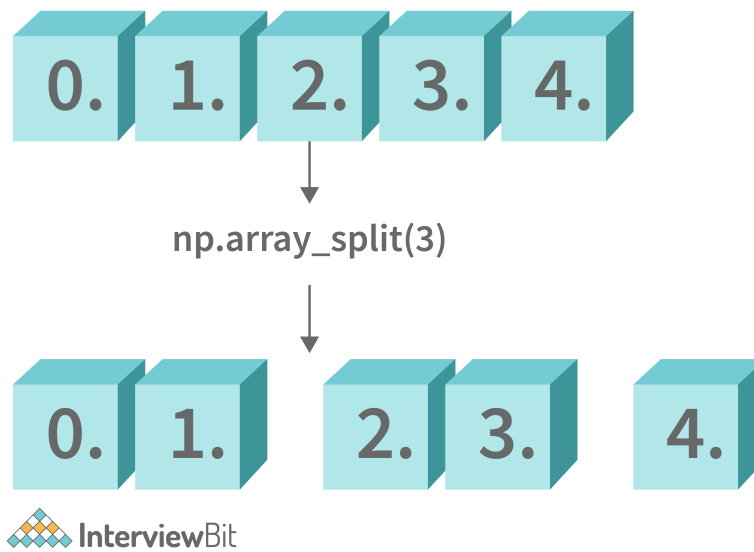
In the above figure, we see there are 5 elements in the array, we want to split the array to 3 subarrays. So L % N = 5%3 = 2 subarrays of size (L//N +1) = (5//3 +1) = 2 are returned and remaining 1 subarray of size L//N = 1 is returned.

Syntax:

```
np.array_split(array, sections, axis=0)
```

where,

- array - Given Input array.
- sections - List of indices or Number of subarrays to be returned.
- axis - Axis along which values have to be appended.

The code for the example illustrated above is:

```
import numpy as np
arr = np.arange(5.0)
split_arrs = np.array_split(arr, 3)
split_arrs
```

Output:

```
[array([0., 1.]), array([2., 3.]), array([4.])]
```

21. How will you implement the moving average for the 1D array in NumPy?

We can make use of the convolve() method. Here, it leverages the way discrete convolution is computed and uses it to find the rolling mean (moving average). Here, the sequence of ones of length equal to the length of the sliding window is convolved with the array.

We first define a calculate_moving_average function which performs the convolution of an array with the sequence of ones of sliding window length w. The mode of the convolve method will be 'valid' to generate the points only where the overlapping of the sequence is complete.

```
import numpy as np
def calculate_moving_average(arr, w):
    return np.convolve(arr, np.ones(w),'valid')/w
```

The above-defined function can be then used for finding the moving average as shown in the examples below:

```
arr1 = np.array([4,5,8,9,3,2,4,2,0,2])
print("Moving average of window length 2: ")
av1 = calculate_moving_average(arr1, 2)
print(av1)

print("Moving average of window length 4: ")
av2 = calculate_moving_average(arr1, 4)
print(av2)
```

Output:

```
Moving average of window length 2:
[4.5 6.5 8.5 6.  2.5 3.  3.  1.  1. ]
Moving average of window length 4:
[6.5  6.25 5.5  4.5  2.75 2.   2.  ]
```

22. How is fliplr different from flipud methods in NumPy?

The fliplr() method is used for flipping an array in the left or right direction. The columns are preserved but the order of elements in the columns would be different than before. This has been represented in the image below:
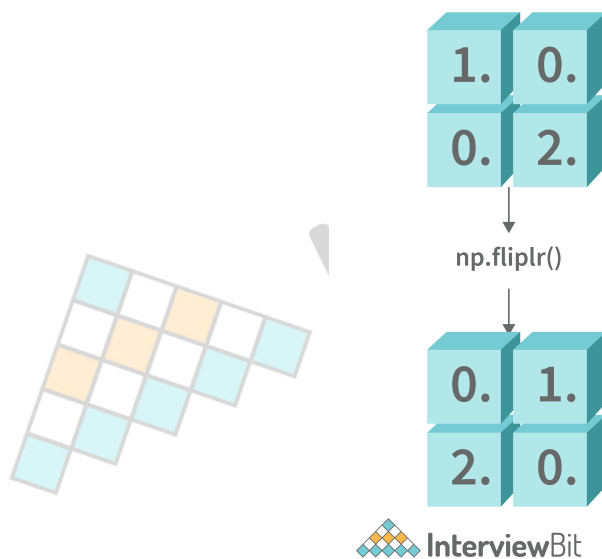

We see that the positions of the elements are flipped left or right than their original position in the result.
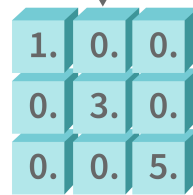
Syntax of fliplr:

```
np.fliplr(arr)
```

where arr is the array that has to be flipped.

The flipud function also flips the array but in the up or down direction. The rows are preserved in this case but they can appear in a different order in the result. This is represented in the image below:
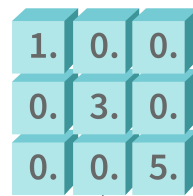
Here, we see that the numbers 1, 3 and 5 elements are flipped in the up/down direction in the result.

The syntax for flipud:

```
np.flipud(arr)
```

where arr is the array that has to be flipped.

NumPy Coding Questions

23. Write a program to convert a string element to uppercase, lowercase, capitalise the first letter, title-case and swapcase of a given NumPy array.

Sample Solution:-

```
import numpy as np

# Create Sample NumPy array
arr = np.array(['i', 'love', 'NumPy', 'AND', 'interviewbit'], dtype=str)

upper_case_arr = np.char.upper(arr)
lower_case_arr = np.char.lower(arr)
capitalize_case_arr = np.char.capitalize(arr)
titlecase_arr = np.char.title(arr)
swapcase_arr = np.char.swapcase(arr)

print("Upper Conversion: ", upper_case_arr)
print("Lower Conversion: ", lower_case_arr)
print("Capitalize First Letter Conversion: ", capitalize_case_arr)
print("Titlecase Conversion: ", titlecase_arr)
print("Swapcase Conversion: ", swapcase_arr)
```

Output:

```
Upper Conversion:  ['I' 'LOVE' 'NUMPY' 'AND' 'INTERVIEWBIT']
Lower Conversion:  ['i' 'love' 'numpy' 'and' 'interviewbit']
Capitalize First Letter Conversion:  ['I' 'Love' 'Numpy' 'And' 'Interviewbit']
Titlecase Conversion:  ['I' 'Love' 'Numpy' 'And' 'Interviewbit']
Swapcase Conversion:  ['I' 'LOVE' 'nUMpY' 'and' 'INTERVIEWBIT']
```

24. Write a program to transform elements of a given string to a numeric string of 10 digits by making all the elements of a given string to a numeric string of 8 digits with zeros on the left.

Sample Solution:-

```
import numpy as np

# Create Sample NumPy array
arr = np.array(['22', '9', '1234', '567', '89102'], dtype=str)

zeroes_filled_arr = np.char.zfill(arr, 8)
print("Transformed array: ")
print(zeroes_filled_arr)
```

Output:

```
Transformed array:
['00000022' '00000009' '00001234' '00000567' '00089102']
```

25. Write a program for inserting space between characters of all elements in a NumPy array.

Sample Solution:-

```
import numpy as np

# Create Sample NumPy Array
arr = np.array(['i', 'love', 'NumPy', 'AND', 'interviewbit'], dtype=str)

transformed_arr = np.char.join(" ", arr)

print("Transformed Array: ")
print(transformed_arr)
```

Output:

```
Transformed Array:
['i' 'l o v e' 'N u m P y' 'A N D' 'i n t e r v i e w b i t']
```

26. Write a program to repeat each of the elements five times for a given array.

Sample Solution:-

```
import numpy as np
# Create Sample NumPy Array
arr = np.array(['i', 'love', 'NumPy', 'AND', 'interviewbit'], dtype=str)

transformed_array = np.char.multiply(arr, 5)
print("Transformed array:")
print(transformed_array)
```

Output:

```
Transformed array:
['iiiii' 'lovelovelovelovelove' 'NumPyNumPyNumPyNumPyNumPy'
 'ANDANDANDANDAND'
 'interviewbitinterviewbitinterviewbitinterviewbitinterviewbit']
```

27. Write a program for creating an integer array with values belonging to the range 10 and 60

```
import numpy as np
arr = np.arange(10, 60)
print(arr)
```

Output:

```
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
 58 59]
```

28. Write a program to add a border of zeros around the existing array.

For example,
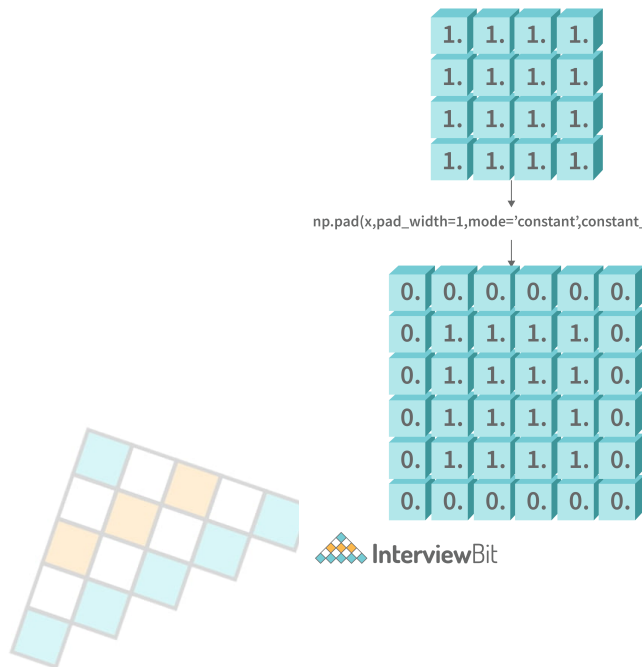If you have the below array:

```
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
```

The resultant array should be: (zeros on the border and 1s within it)

```
[[ 0.  0.  0.  0.  0.  0.]
 [ 0.  1.  1.  1.  1.  0.]
 [ 0.  1.  1.  1.  1.  0.]
 [ 0.  1.  1.  1.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.]]
```

Solution:-

This can be achieved by using the pad method of the NumPy library.


```
import numpy as np

# Create NumPy arrays filled with ones
ones_arr = np.ones((4,4))

print("Transformed array:")
transformed_array = np.pad(ones_arr, pad_width=1, mode='constant', constant_values=0)
print(transformed_array)
```

Output:

```
Transformed array:
[[0. 0. 0. 0. 0. 0.]
 [0. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0.]]
```

29. Write a program for changing the dimension of a NumPy array.

We can achieve this by overriding the shape attribute of the NumPy array.

Sample Solution:

```
import numpy as np

#Create NumPy array
arr = np.array([1,2,3,4,5,6,7,8,9])
print("Original Shape: ", arr.shape)

# Change the shape/dimension of the array
arr.shape = (3, 3)
print("Transformed Matrix :")
print(arr)
print("Transformed Shape: ",arr.shape)
```

Output:

```
Original Shape:  (9,)
Transformed Matrix :
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Transformed Shape:  (3, 3)
```

In this approach, care has to be taken w.r.t the number of elements present in the original array before changing the dimensions. Otherwise, it will result in the ValueError as shown below:

```
import numpy as np

# We have array of 8 elements
arr = np.array([1,2,3,4,5,6,7,8])

# We are trying to convert the 1D array to a 3D array which expects 9 elements
arr.shape = (3, 3)
print(arr)
```
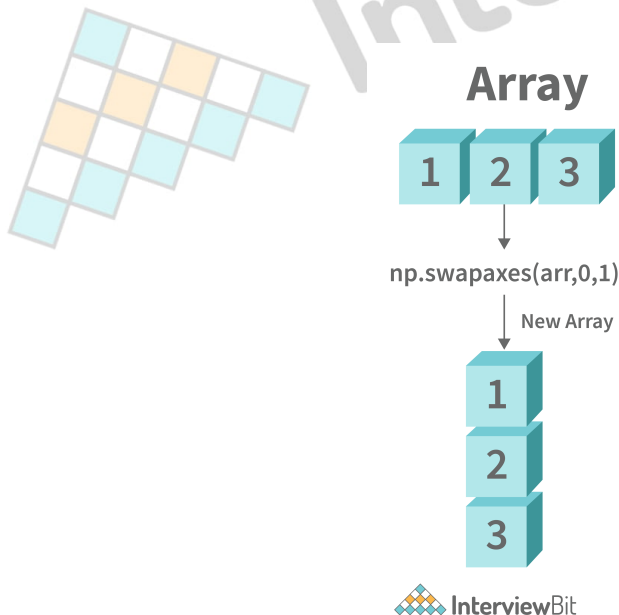
Running this code would result in:

```
      1 import numpy as np
      2 arr = np.array([1,2,3,4,5,6,7,8])
----> 3 arr.shape = (3, 3)
      4 print(arr)

  ValueError: cannot reshape array of size 8 into shape (3,3)
```

30. Write a program for interchanging two axes of the NumPy array.

This can be achieved by using the swapaxes method of NumPy. The below image illustrates the meaning of swapping axes.


```
import numpy as np
arr = np.array([[1,2,3]])
print("Original array: ")
print(arr)

#Swap axes
axis_swapped_arr =  np.swapaxes(arr,0,1)
print("Transformed array: ")
print(axis_swapped_arr)
```